ISCA Archive
http://www.isca-speech.org/archive

ITRW on Nonlinear Speech Processing (NOLISP 07)
Paris, France
May 22-25, 2007

Efficient Viterbi algorithms for lexical tree based models

S. España-Boquera, M.J. Castro-Bleda, F. Zamora-Martínez, J. Gorbe-Moya

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
`sespana@dsic.upv.es`

Abstract

In this paper we propose a family of Viterbi algorithms specialized for lexical tree based FSA and HMM acoustic models. Two algorithms to decode a tree lexicon with left-to-right models with or without skips and other algorithm which takes a directed acyclic graph as input and performs error correcting decoding are presented. They store the set of active states topologically sorted in contiguous memory queues. The number of basic operations needed to update each hypothesis is reduced and also more locality in memory is obtained reducing the expected number of cache misses and achieving a speed-up over other implementations.

1. Introduction

Most of large vocabulary Viterbi based recognizers (for speech, handwritten or other recognition tasks, although speech terminology is used in this work with no loss of generality) make use of a lexicon tree organization which has many advantages over a linear lexicon representation [1, 2]. As it is shown in the literature, more compact representations are possible (using a *lexicon network* [3], which is a minimized Finite State Automaton –FSA–) but the gain in space is accompanied with a more complex Viterbi decoder. Therefore, lexical tree organization is a very good tradeoff between compact space representation and adequacy for decoding.

The search space in a recognizer can be huge and the key to achieve practical performance is to consider only the set of active hypothesis (those with non trivial zero probability) and to apply pruning techniques such as beam search which only maintain active the best hypothesis.

Large vocabulary one-step decoders [4] usually keep a set of lexical tree based Viterbi parsers in parallel. Two common approaches are the *time-start copies* and *language model history copies* [5, 6]. In the time-start approach, all hypothesis competing in a tree parsing share the same word start time. When a trigram language model is used, the language model history copies approach maintains a tree parsing for every bigram history $(w1, w2)$. This second approach has a loss of optimality which is known as *word-pair approximation* [6]. In both cases, it is straightforward to use a specialized Viterbi algorithm for the lexical tree model and, as the core of an automatic speech recognizer lies in the search process, every little improvement in performing specialized decoding of lexical tree models has a great impact in the overall performance. Therefore, it is not strange to find specialized algorithms which take advantage of the properties of tree based HMM models which integrate the tree lexicon and the acoustic HMM models.

This work has been partially supported by the Spanish Government (TIN2006-12767) and by the Generalitat Valenciana (GVA06/302).

When the acoustic models are strict left-to-right without skips, the resulting expanded HMM model is acyclic if loops are ignored, and every node has only zero or one preceding state to take into account in the dynamic programming equation. Left-to-right units with skips are known as Bakis topology and have a widespread use as acoustic models in most recognizers. When those models are used in conjunction with a tree lexicon, the number of predecessors given an active state can be zero, one or two.

Not expanding the acoustic models in the tree lexicon and maintaining a pure tree structure which matches a phone-graph is another possibility. In this case, the input is no longer a sequence of acoustic frames but a phone-graph (a directed acyclic graph –DAG– labelled with phones and acoustic scores). Besides the capability of using a directed acyclic input, the possibility of insertions, deletions and substitutions of phones is needed to tolerate the errors in the phone-graph generation.

In this work, three specialized Viterbi algorithms based on contiguous memory queues (FIFO data structures) are proposed. When performing a Viterbi step, a new result queue is created with the help of one or several auxiliary queues.

The basic algorithm uses left-to-right HMM acoustic models with no skips. This algorithm can be applied whenever acoustic left-to-right models without skips are used: it can be used for isolated word or continuous speech recognition, either with a one-step or a two-step approach, with time-start or language model history copies, and also within-word or across-word context dependent units (triphones, quinphones, etc.). A simple extension is presented to show how to use it with across-word context dependent models [7].

A second version of the algorithm extends the first one to allow the use of skips in the acoustic units with a negligible additional cost.

The last proposed algorithm performs an error correcting Viterbi decoding and is capable of analyzing a DAG instead of a sequence. This algorithm can be used, for instance, to obtain a word-graph from a phone-graph.

2. Left-to-right without skips algorithm

If a lexicon tree is expanded with left-to-right acoustic HMM models without skips, the following observations about the expanded tree models are straightforward:

- Every state has at most two predecessors: itself and possibly his parent.
- If we ignore the loops, the expanded model is acyclic. Therefore, a topological order is possible in general.
- A level traversal of the tree provides a topological order with some additional features:

- The children of a given node occupy contiguous positions. The grandchildren also occupy contiguous positions.
- If a subset of states is stored in topological order and we generate the children of every active state following that order, the resulting list also is ordered with respect to the topological order.

2.1. Model representation

A tree model T of n states is represented with three vectors of size n and one of size $n+1$ as follows:

- $loop\_prob$ stores the loop transition probabilities.
- $from\_prob$ stores the parent incoming transition probabilities.
- e_index stores the index of the associated emission probability class associated to the acoustic frame to be observed. The vector of emission probabilities can be obtained with a multilayer perceptron in a hybrid model [8] or with a set of mixture of Gaussian distributions in a conventional continuous density HMM.
- $first\_child$ stores the index of the first child. The last child is deduced by looking the first child of the next state thanks to the topological sorting. This representation allows specifying an empty set of children. A sentinel in position $n+1$ is needed for the last state.

2.2. Viterbi-Merge algorithm

The Viterbi-M algorithm takes a sequence of acoustic frames as input and updates a set of active states after observing every frame (a Viterbi step). An active state is composed by an index state and a score (i, s). A queue $\alpha(t)$ (a FIFO data structure) is used to store the set of active states at time t. The purpose of a Viterbi step consists of creating another queue $\alpha(t+1)$ given the model T, the current queue $\alpha(t)$ and the vector of emission probabilities $emission$ associated to the observed acoustic frame.

An auxiliary queue aux_child is used to store temporally the scores of states produced by the transitions from parent to child. The algorithm proceeds as follows (see Figure 1):

1. $best\_prob \leftarrow 0$.
2. For every active state (i, s) of the queue $\alpha(t)$ whose score s is above the beam threshold:
   (a) $s\_next \leftarrow s \cdot loop\_prob[i]$.
   (b) While the first active state (i', s') of the queue aux_child satisfies $i' < i$, extract it and place $(i', s' \cdot emission[e\_index[i']])$ in $\alpha(t+1)$.
   (c) If the first active state (i', s') of the queue aux_child satisfies $i' = i$, drop it and update the score $s\_next \leftarrow \max(s\_next, s')$.
   (d) $s\_next \leftarrow s\_next \cdot emission[e\_index[i]]$, insert (i, s_next) in the queue $\alpha(t+1)$ and update $best\_prob \leftarrow \max(best\_prob, s\_next)$.
   (e) For every state j from $first\_child[i]$ to $first\_child[i+1]-1$, add $(j, s \cdot from\_prob[j])$ to the queue aux_child.
3. For every active state (i', s') of the queue aux_child, extract it and place $(i', s' \cdot emission[e\_index[i']])$ in the queue $\alpha(t+1)$.

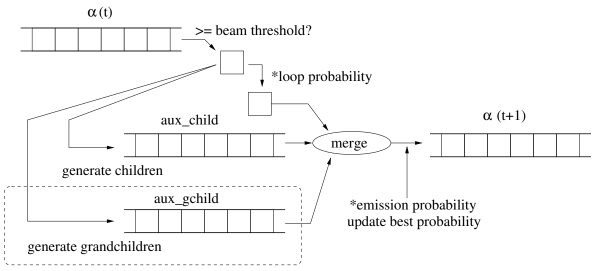

Figure 1: *Viterbi-M and Viterbi-MS algorithms. The queue* aux_gchild *(dotted part) is only used in Viterbi-MS.*

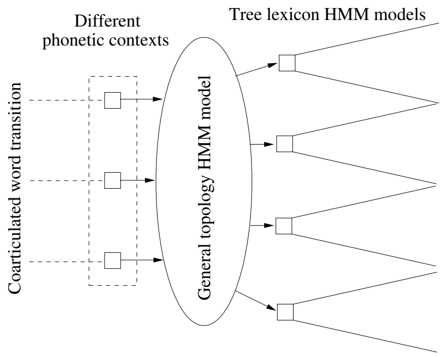
Different phonetic contexts
Tree lexicon HMM models
Coarticulated word transition
General topology HMM model

Figure 2: *HMM model for across-word context dependent units. The general topology HMM model represents only the first context dependent phones of the words.*

The active states of $\alpha(t)$ whose score is below the beam threshold are discarded. Whenever an active state is placed in the queue $\alpha(t+1)$, the emission probability associated to it is applied, and the best probability is updated. This value is used to obtain the beam threshold for the following Viterbi step.

This algorithm is linear with the number of active states and the number of children of active states which is an upper bound to the number of active states in the resulting queue. In total, the global cost is linear with the number of active states, which is the same as any reasonable implementation of a conventional Viterbi implementation. The main advantage of this algorithm is the use of contiguous memory FIFO queues to store the active states. Therefore, a better cache performance and an internal loop with less overhead is obtained compared to other algorithms that use linked lists or use hash tables to store and look up the set of active states. Therefore, the asymptotic cost is the same but a practical speed-up is obtained.

2.3. Extension to across-word context dependent units

Since this algorithm is used on lexical tree models with expanded acoustic HMM models, the use of context dependent units is straightforward for within-word context modeling.

Across-word models consider a different context dependent unit at the beginning of a word to take into account the last phones of the preceding word during continuous speech recognition. It would be very inefficient to use a different tree model for every possible context since they only differ in the first context dependent acoustic models. Therefore, a model which resembles a tree lexicon excepting the root is used. This model can be composed of two models: a general HMM connecting a set of tree lexicon models (see Figure 2).

A set of trees can be traversed by levels as if they were just one tree and the resulting model can be used with the same algorithm with no modification. Therefore, a conventional Viterbi algorithm can be used to update the scores of the states of the general topology HMM part of the model, and the rest of the model (a forest) can be computed with the Viterbi-M algorithm.

3. Left-to-right with skips algorithm

This algorithm generalizes the previous one by allowing the use of Bakis HMM acoustic models.

3.1. Model representation

The model representation is similar to the previous section. The only difference is another vector $skip\_prob$ which stores, for every state, the incoming skip transition probabilities. In order to iterate over the set of grandchildren of a given state i, the algorithm loops from $first\_child[first\_child[i]]$ to $first\_child[first\_child[i + 1]] - 1$.

3.2. Viterbi-Merge algorithm with skips

The Viterbi-MS algorithm is the same of previous section but another auxiliary queue *aux_gchild* is used to store the active states with scores computed by means of the skip transitions. Every time an active state is extracted from $\alpha(t)$, the set of grandchildren is used to add items to the queue *aux_gchild* just as the set of children is used to add items to the other auxiliary queue. Now, the resulting queue $\alpha(t + 1)$ is obtained by merging the loop transition score of the processed active state with the states from the two auxiliary queues (see Figure 1).

This algorithm is linear with the number of active states and the number of children and grandchildren of active states. The resulting cost is thus linear with the number of active states.

3.3. Extension to across-word context dependent units

The same observations of previous algorithm are also applicable here.

4. Error-Correcting Viterbi for DAGs

The last proposed algorithm performs an error correcting Viterbi decoding and is capable of analyzing a DAG instead of a sequence.

4.1. Model representation

A tree model T of n states where symbols are placed at the transitions is represented with two vectors of size n and other of size $n + 1$ as follows:

- $symbol$ stores the incoming transition label.
- $from\_prob$ stores the incoming transition probability.
- $first\_child$ stores the index of the first child as in the previous algorithms.

A table with the costs of insertions, deletions and substitution of every symbol is also required.

4.2. Error-Correcting Viterbi-Merge algorithm for DAGs

The Viterbi-MEC-DAG algorithm takes a DAG as input. Consider the phone-graph of Figure 3. A set of active states is associated to every vertex of the input DAG. The algorithm applies two different procedures associated to the input DAG,

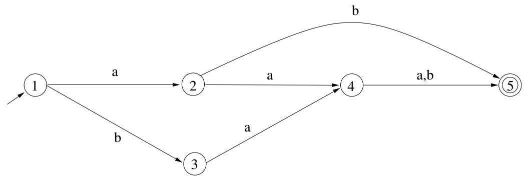
$$\begin{tikzcd} 1 \arrow[r, "a"] \arrow[dr, "b"'] & 2 \arrow[r, "a"] \arrow[rr, bend left, "b"] & 4 \arrow[r, "a,b"] & 5 \\ & 3 \arrow[ur, "a"'] & & \end{tikzcd}$$

Figure 3: *Phone-graph example.*

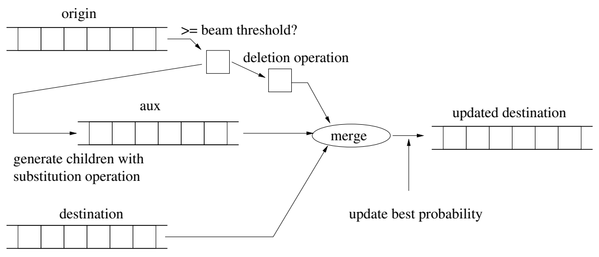

Figure 4: *Viterbi-MEC-DAG edge-step procedure.*

which must be applied following the DAG topological order: the vertex-step procedure must be applied to every vertex before using this set of active states as the origin of an edge edge-step procedure.

4.2.1. Edge-step

For every edge, a Viterbi step takes the set of active states of the origin vertex and use them to update the active states of the destination vertex. This procedure only considers the cost of deletions and substitutions. As can be observed in Figure 4, this algorithm is similar to the Viterbi-M algorithm where loop probability updating is replaced by the deletion operation, the generation of children states corresponds to the substitution operation (including a symbol by itself or a correct transition). Another difference, which can be also used in Viterbi-M and Viterbi-MS to process a DAG as input data, is the presence of second input queue which stores the active states already updated at the destination vertex by means of other edges of the DAG. These values are simply merged and this queue is not needed when the input data is a sequence. The cost of this procedure is linear with the number of active states in both input queues because the number of generated successor states grows linearly with the number of active states.

4.2.2. Vertex-step

Once all edges arriving at a given vertex have been processed, the insertion operation is considered. This operation updates a set of active states without consuming any symbol. As can be observed in Figure 5, the output of the auxiliary queue is used to insert more active states in the same queue to take into account the possibility of several insertion operations. The cost is not linear with the number of active states: a sole active state at the root could, in principle, activate all the states of the model, but most of them are expected to be pruned by the beam search depending on the cost of insertions and the beam width. The cost of this operation is linear with the number of active states before applying the procedure plus the number of active states after the procedure, which is bounded by the number of states in the model.

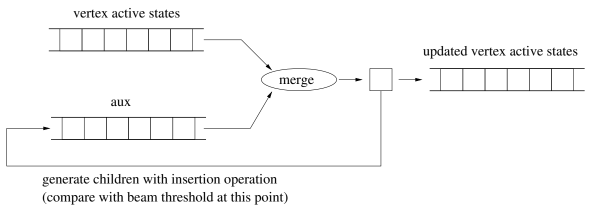
vertex active states
aux
merge
updated vertex active states
generate children with insertion operation
(compare with beam threshold at this point)

Figure 5: *Viterbi-MEC-DAG vertex-step procedure.*

Num. states	Hash swap	A. Envelope	Viterbi-M
9 571	3.001	16.082	29.350
76 189	2.761	12.924	28.036
310 888	1.922	6.442	24.534

Table 1: *Experimental results. The results are shown in millions of active states or hypothesis updated per second.*

5. Experimental results

A previous work related to lexical tree Viterbi decoding we were aware of after our algorithms were developed is the *active envelope* algorithm [9]. This algorithm also uses a total order which subsumes the partial order of the states of the model and places siblings contiguously. This algorithm is specified for left-to-right models without skips, so it is only comparable to our first algorithm Viterbi-M. The active envelope algorithm uses a single linked list to perform a Viterbi step, which is an advantage in memory usage. Since this algorithm modifies the original set of active states, it is restricted to sequential input data and cannot be used when the input data is a DAG. In order to use only a list, the active hypothesis in active envelope algorithm are traversed in *reverse* topological order. The "price to pay" for this advantage is the need of linked lists instead of contiguous memory arrays. Since the use of linked lists cannot assure memory locality and the cost of traversing them is greater than traversing memory arrays, it is expected to perform worse than Viterbi-M algorithm. The memory occupied by an active hypothesis is an index state and a score; if linked lists are used, a pointer is also needed: so a linked list needs approximately 50% or 100% more memory per active state depending on the computer architecture. On the other hand, the use of memory arrays needs an estimation of the number of active states.

In order to compare the performance of our Viterbi-M algorithm, two more algorithms have been implemented: a conventional Viterbi algorithm based on hash tables with chaining to store and to look up the active states and the active envelope algorithm. All algorithms have been implemented in C++ and use the same data structures to represent the tree based HMM models as described in section 2.1.

The experiments were done on a Pentium D machine at 3GHz with 2 Gbytes of RAM using a Linux with kernel `2.6.18` and the `gcc` compiler version `4.1.2` with `-O3` optimization. The lexical trees used in the experiments were obtained by expanding 3-state left-to-right without skips hybrid neural/HMM acoustic models in the tree lexicon. The size of these trees varies from 9 571 to 310 888 states. Only the Viterbi decoding time has been measured (the emission scores calculation and other preprocessing steps were not taken into account). The result is shown in Table 1. The speed is measured in millions of active states updated per second.

6. Conclusions

In this paper, three Viterbi algorithms specialized for lexical tree based FSA and HMM acoustic models have been described. Two of these algorithms are useful to decode a set of words given a sequence of acoustic frames and the third one is useful to parse a phone-graph with error-correcting edition operations. These algorithms are based on contiguous memory queues which contain the set of active states topologically sorted.

Although the asymptotic cost of these algorithms is the same as any reasonable implementation of the Viterbi algorithm, the experimental comparison between the Viterbi-M algorithm, a conventional Hash-table swapping algorithm and the active envelope algorithm, shows that our algorithm is approximately 10 times faster than the hash-table swapping implementation and from 2 to 4 times faster than the active envelope algorithm. A decrease in speed with the size of the models is observed in the three algorithms, which is possibly related with the main memory and the cache relative speeds. For this reason, more experimentation is needed in order to better understand this behaviour and also to study the effect of other parameters such as the beam width of the pruning during the search.

7. References

[1] J. Klovstad and L. Mondshein, "The CASPERS linguistic analysis system," *IEEE Transactions on Acoustic, Speech and Signal Processing*, vol. 23, no. 1, pp. 118–123, Feb. 1975.

[2] D. Klatt, *Trends in Speech Recognition*. Prentice-Hall, 1980, ch. Scriber and Lafs: Two New Approaches to Speech Analysis, pp. 529–525.

[3] K. Demuynck, J. Duchateau, and D. V. Compernolle, "A Static Lexicon Network Representation for Cross-word Context Dependent Phones," in *Proc. European Conference on Speech Communication and Technology*, vol. I, Rhodes, Greece, September 1997, pp. 143–146.

[4] X. L. Aubert, "An overview of decoding techniques for large vocabulary continuous speech recognition," in *Computer Speech and Language*, vol. 16, 2002, pp. 89–114.

[5] S. Ortmanns, H. Ney, F. Seide, and I. Lindam, "A comparison of time conditioned and word conditioned search techniques for large vocabulary speech recognition," in *Proc. ICSLP '96*, vol. 4, Philadelphia, PA, 1996, pp. 2091–2094.

[6] H. Ney and S. Ortmanns, "Dynamic programming search for continuous speech recognition," *IEEE Signal Processing Magazine*, vol. 16, no. 5, pp. 64–83, 1999.

[7] S. Kanthak, A. Sixtus, S. Molau, and H. Ney, "Within-word vs. across-word decoding for online speech recognition," 2000.

[8] Y. Konig, H. Bourlard, and N. Morgan, "REMAP: Recursive estimation and maximization of A posteriori probabilities — application to transition-based connectionist speech recognition," in *Advances in Neural Information Processing Systems*, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds., vol. 8. The MIT Press, 1996, pp. 388–394.

[9] P. Nguyen, L. Rigazio, and J.-C. Junqua, "EWAVES: An efficient decoding algorithm for lexical tree based speech recognition," in *ICSLP-2000*, vol. 4, 2000, pp. 286–289.